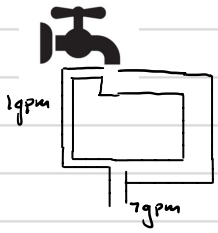
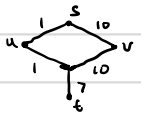
Max Flow (prep) Cormen §26

Think: pipes + H_2O

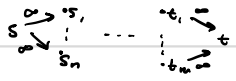


=



Extensions

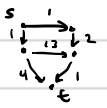
- mult source/sink



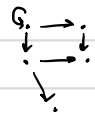
- Backwards edges

Def'n A flow network consists of (Not strictly required: intro pseudo nodes)

- A directed graph $G=(V,E)$ s.t. a) if $(u,v)\in E$, then $(v,u)\notin E$ b) No self-loops
- A non-negative capacity func $c: E \to \mathbb{R}_{\geq 0}$ (i.e. $\forall (u,v)\in E,\ c(u,v)\geq 0$). $(u,v)\notin E \Rightarrow c(u,v)=0$
- Two distinguished vertices: a source s, and a sink t
- Every vertex lies on some path from source to sink



✓



✗

Def'n A flow is a function $f: V\times V \to \mathbb{R}$ s.t.

1. (capacity constraint) $\forall u,v\in V,\quad 0\leq f(u,v)\leq c(u,v)$ (flow can't exceed capacity)
2. (flow conservation) $\forall u\in V-\{s,t\}\quad \sum_{v\in V} f(v,u) = \sum_{v\in V} f(u,v)$ (no leaky pipes)

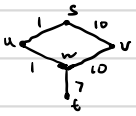
total flow into u ↑ ↑ total flow out of u

after def'n, eg of appl.

Def'n The Value of a flow f is $|f| := \sum_{v\in V} f(s,v) - \sum_{v\in V} f(v,s)$

flow out of source − flow into source

So the Max Flow problem is: given a flow network, find a flow f w/ maximal value $|f|$.



edges	c
(s,u)	1
(s,v)	10
(u,w)	1
(v,w)	10
(w,t)	7

edges	f
(s,u)	1
(s,v)	1
(u,w)	6
(v,w)	6
(w,t)	7

$|f|=7$ Note: maximal value is unique but many flows can attain this value

Ford Fulkerson Method/Alg (FFA)

Idea is as follows

FFA(G, s, t): ← flow network

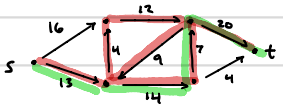
Initialize f to all 0's

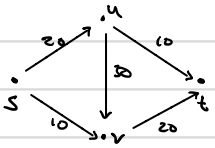
while ∃ path from s to t w/ spare capacity (augmenting path)

use that capacity (add it to the flow) ← make sure it doesn't violate the def'n of a flow

return f

How do we go about this so we don't get "stuck" @ a suboptimal soln? eg





we need a way to keep track of how much capacity is left, (residual)

as well as how much mass was moved (esp. if we need to send it back)

Def'n (Residual network) any flow

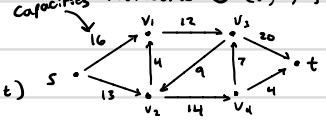
Given a flow network $G=(V,E,c,s,t)$ & flow f, the residual network $G_f=(V, E_f, c_f, s, t)$

Where $\forall u,v\in V$

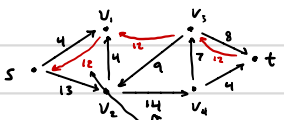
$$c_f(u,v)=\begin{cases} c(u,v)-f(u,v) & \text{if } (u,v)\in E \quad \text{how much cap. is left} \\ f(v,u) & \text{if } (v,u)\in E \quad \text{how much mass was moved} \\ 0 & \text{o.w.} \end{cases}$$

and $E_f = \{(u,v)\in V\times V \text{ s.t. } c_f(u,v)>0\}$

capacities Network $G=(V,E,c,s,t)$



Residual network



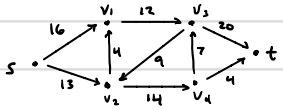
residual capacities c_f

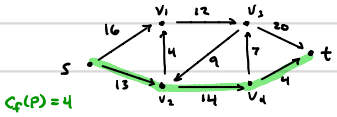
vertex	Flow
(s,v_1)	12
(v_1,v_3)	12
(v_3,t)	12

Example

Network G | Residual network G_f

$f = 0$, $|f| = 0$

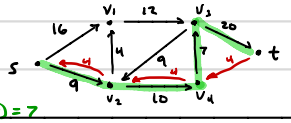




$c_f(P) = 4$

$f(s,v_2) = 4$, $f(v_2,v_4) = 4$, $f(v_4,t) = 4$

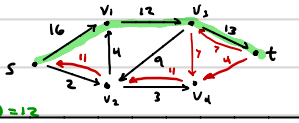
$|f| = 4$



$c_f(P) = 7$

$f(s,v_3) = 11$, $f(v_2,v_4) = 11$, $f(v_4,t) = 4$
$f(v_4,v_3) = 7$, $f(v_3,t) = 7$

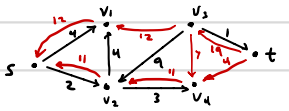
$|f| = 11$



$c_f(P) = 12$

$f(s,v_3) = 11$, $f(v_2,v_4) = 11$, $f(v_4,t) = 4$
$f(v_4,v_3) = 7$, $f(v_3,t) = 12$, $f(s,v_1) = 12$
$f(v_1,v_3) = 12$

$|f| = 23$



Find an augmenting path: a path in G_f from s to t.

Note can move as much mass on such a path as the min residual capacity on all its edges.

$c_f(P) := \min\{c_f(u,v) : (u,v) \text{ on } p\}$ ← bottle neck

FFA:

Initialize f to all 0's

while $\exists$ path from s to t w/ spare capacity

use that capacity (add it to the flow)

return f

Notes on performance:

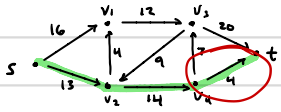
1. Flow increases by $c_f(P)$ after each step of while loop
2. Performance depends on how you find augmenting paths
   - 2.a. Some choices can be rigged to cause alg to not terminate (wiki has good eg)
   - 2.b. For integer capacities, running time is $O(E \cdot f^*)$ since incr. by ≥ 1 unit each while loop (max # edges in a path; max flow value)
     Mention integer capacities ⇒ integer flow (on edges)
   - 2.c. Edmonds-Karp algorithm is FF + BFS for augmenting paths
     - 2.c.i. $O(VE^2)$ (Keep track of how many times any edge $(u,v) \in E$ achieves $c_f(P) = c_f(u,v)$ for aug. path p) See Cormen pg 725

So that's the running time, what about proof of correctness?

Graph Cuts (Cormen §26.2)

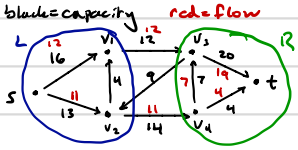
Observe: Whenever we find an aug. path, we're basically "cutting out" one of the edges

When can we no longer increase flow? When can't find an aug. path: all $s \to t$ paths have been "cut out." This leads to



Def'n (cut)

A cut of a network $G = (V,E)$ is a partition of the vertices into two sets L, R s.t. $L \cap R = \emptyset$, $L \cup R = V$.



The edges that cross the cut are particularly interesting: "cut them out" & graph becomes disconnected.

Two quantities of concern are the flow across the cut, & the capacity across it

Def'n The net flow across a cut (L,R) is given by $f(L,R) = \sum_{u \in L} \sum_{v \in R} f(u,v) - \sum_{u \in L} \sum_{v \in R} f(v,u)$

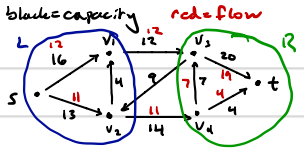
$f(L,R) = (12 + 11) - (0) = 23$

Def'n An (S,t)-cut is a cut (L,R) of a flow network s.t. $s \in L$ and $t \in R$.

Def'n The capacity of a cut (L,R) is given by $c(L,R) = \sum_{u \in L} \sum_{v \in R} c(u,v)$

$c(L,R) = 12 + 14 = 26$

Def'n A min cut is a cut whose capacity is the smallest over all cuts



$f(L,R) = (12 + 11) - (0) = 23$

$C(L,R) = 12 + 14 = 26$

Lemma 26.4 for any cut (L,R), for any flow f, the net flow across the cut is the flow value. i.e.

$$f(L,R) = |f|$$

Proof:

$$|f| = \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) \quad \text{by def'n}$$

$$= \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in L - \{s\}} \left(\sum_{v \in V} f(u,v) - \sum_{v \in V} f(v,u) \right) \quad \text{by "no leaky pipes" this is a fancy way to write 0}$$

$$= \sum_{v \in V} f(s,v) - \sum_{v \in V} f(v,s) + \sum_{u \in L - \{s\}} \sum_{v \in V} f(u,v) - \sum_{u \in L - \{s\}} \sum_{v \in V} f(v,u) \quad \text{expand}$$

$$= \sum_{v \in V} \left(f(s,v) + \sum_{u \in L - \{s\}} f(u,v) \right) - \sum_{v \in V} \left(f(v,s) + \sum_{u \in L - \{s\}} f(v,u) \right) \quad \text{rearrange}$$

$$= \sum_{v \in V} \sum_{u \in L} f(u,v) - \sum_{v \in V} \sum_{u \in L} f(v,u) \quad \text{combine}$$

$$= \sum_{v \in L} \sum_{u \in L} f(u,v) + \sum_{v \in R} \sum_{u \in L} f(u,v) - \sum_{v \in L} \sum_{u \in L} f(v,u) - \sum_{v \in R} \sum_{u \in L} f(v,u) \quad V = L \cup R \text{ and } L \cap R = \emptyset \text{ so partition } V$$

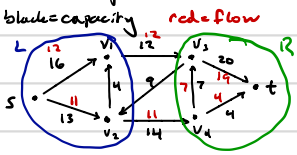
$$= \sum_{v \in R} \sum_{u \in L} f(u,v) - \sum_{v \in R} \sum_{u \in L} f(v,u) + \left(\sum_{v \in L} \sum_{u \in L} f(u,v) - \sum_{v \in L} \sum_{u \in L} f(v,u) \right) \quad \text{(the bracketed part} = 0\text{)}$$

$$= \sum_{v \in R} \sum_{u \in L} f(u,v) - \sum_{v \in R} \sum_{u \in L} f(v,u)$$

$$= f(S,T)$$

We can use this to show that the capacity of any cut gives an upper bound on how large the value of any flow can be

Corollary 26.5 For any cut (L,R) and any flow f, $|f| \le C(L,R)$



$|f| = 12 + 11 = 23 \le 12 + 14 = 26 = C(L,R)$

Proof:

$|f| = f(S,T)$ — Lemma 26.4

$$= \sum_{u \in L} \sum_{v \in R} f(u,v) - \sum_{u \in L} \sum_{v \in R} f(v,u) \quad \text{def'n of net flow across a cut}$$

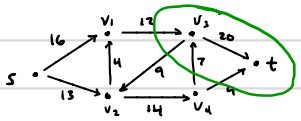
$$\le \sum_{u \in L} \sum_{v \in R} f(u,v) \quad \text{drop "−" terms}$$

$$\le \sum_{u \in L} \sum_{v \in R} c(u,v) \quad \text{flow} \le \text{capacity across every edge}$$

$$= C(L,R) \quad \text{def'n}$$

This then allows us to prove that a terminating FFA results in a maximal flow

Thm 26.6 (Min Cut Max Flow thm)



$C(L,R) = 12 + 7 + 4 = 23$

TFAE: 1) f is a max flow in G

2) G_f has no aug flows

3) $|f| = C(L,R)$ for some cut (L,R)

Proof: we will show $1 \Rightarrow 2 \Rightarrow 3 \Rightarrow 1$ so $1 \Leftrightarrow 2 \Leftrightarrow 3$

$1 \Rightarrow 2$) is oac, say f is a maximal flow & p is an aug. path., Since aug f w/ p would increase flow, f wasn't maximal ☒

($2 \Rightarrow 3$) Say $\nexists$ an augmenting path in G_f. Let

$L = \{u \in V : u \text{ is reachable from } s\}$ $\quad R = V - L$.

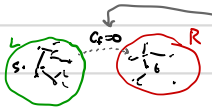
Then $|f| = f(L,R)$ by lemma 26.5

$$= \sum_{u \in L} \sum_{v \in R} f(u,v) - \sum_{u \in L} \sum_{v \in R} f(v,u)$$

$f(u,v) = c(u,v)$ by Ⓐ; $f(v,u) = 0$ by Ⓑ

$$= \sum_{u \in L} \sum_{v \in R} c(u,v) = C(L,R)$$

$$c_f(v,r) = \begin{cases} c(v,r) - f(v,r) & : f(v,r) \in E \\ f(r,v) & : f(r,v) \in E \\ 0 & \text{o.w.} \end{cases}$$



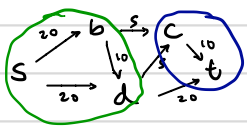
By def'n of L (everything reachable from s in G_f), we can't have $c_f(u,v) > 0$ (an edge across the cut) for $u \in L$, $v \in R$, since then $v \in L \cap R$ ☒ Hence $c_f(v,r) = 0 \ \forall u \in L, r \in R$

Ⓐ In particular, $c(v,r) - f(v,r) = 0$ i.e. $c(v,r) = f(v,r)$

Ⓑ also $f(r,v) = 0$ too

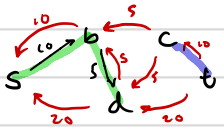
$3 \Rightarrow 1$) Cor 26.5 $\Rightarrow |f| \le C(L,R)$ for all cuts (L,R) so $|f| = C(L,R) \Rightarrow |f|$ is maximal

eg Find the min cut of the following flow network, given the max flow

$G =$ 

(u,v)	$f(u,v)$
(s,b)	10
(s,d)	20
(b,c)	5
(b,d)	5
(d,c)	5
(d,t)	20
(c,t)	10

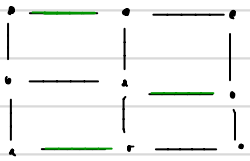
$|f| = 10+20 = 30$

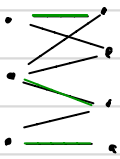
G_f 

$L = \{s, b, d\}$ $R = \{c, t\}$ $C(L,R) = 20+5+5 = 30$

Appl. of Min Cut / Max Flow: Maximum bipartite matching

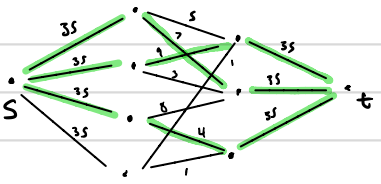
Def'n a matching M of a graph $G=(V,E)$ is an edge subset $M \subseteq E$ s.t.
no two edges in M have any vertices in common.





Def'n a maximal matching M is a matching s.t. for any other matching M',
$|M| \geq |M'|$.

For a weighted graph, a maximal weighted matching M is one s.t. $\sum_{e \in M} w(e) \geq \sum_{e \in M'} w(e)$



(Kidney donors, match making, etc)

$9+7+4 = 20$

Run FFA to find optimal matching

Possible: red-black trees, B trees, etc.

Greedy algorithms Dasgupta §5

An algorithmic approach where soln's are built up piece-by-piece, where each piece "looks the best" at that time.

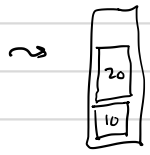
Typically in optimization where the prob. is cast as a seq. of choices, each choice chosen to provide the most improvement possible. Hoping/proving that @ the end you get to a globally optimal soln.

While the greedy heuristic can be applied to most optimization problems, we will see that we need 3 key ingredients to show that it leads to an optimal soln

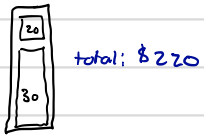
First: Cast the problem as one where a seq. of choices are made, and each choice leaves a single sub-problem to solve.

eg 0-1 Knapsack problem

Thief, has backpack w/ limited capacity
a set of items w/ certain weights & values
How to maximize value?
Naïve approach: pick highest value/lb



Can we do better?



other soln:

So greedy gave the worst soln

So the greedy approach failed. Need another ingredient to make the greedy approach work

Second: Greedy choice property: Prove ∃ optimal soln that makes the greedy choice

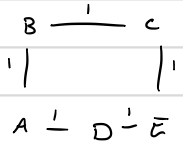
Note: 0-1 Knapsack problem doesn't have ↖ hence why the greedy approach failed

Note: greedy choice property shows greedy choice is "safe" (part of optimal soln) but does not guarantee that making a sequence of greedy choices leads to optimal soln.

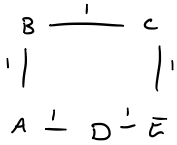
eg Maximum weighted path

graph w/ edge weights. Given 2 vertices, find a path w/ maximum total weight between them (no edge repeated)

Say



MaxPath(A,C) = 3 (via A-D-E-C)



Say greedy heuristic is "add maximum weighted edge" if edge untraversed, until you hit target node.

Will definitely fail, but it does satisfy our first 2 ingredients: Seq of choices (one subproblem), and optimal Sol'n contains greedy choice.

3 A-D-E-C 4 A-B-C-E-D 3 D-A-B-C

But! Observe: $MaxPath(A,C) \neq MaxPath(A,D) + MaxPath(D,C)$

So the sub-problem we chose ("pick a weighty edge") cannot be combined into an optimal soln.
i.e. problem can't be broken down to greedy subproblems that can be assembled to an optimal soln,
This problem lacks optimal substructure, the last ingredient

Third: Optimal Substructure: A problem has O.S. if an optimal soln to a problem contains within it optimal solutions to sub-problems.

In Summary, to guarantee that a greedy approach works, need to:

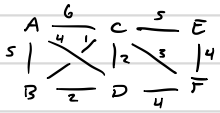
Cast as one-subproblem choices →	1. Cast as series of choices, each choice is one sub-problem to choose
Show greedy is a safe choice →	2. Prove ∃ optimal soln that makes the greedy choice
demonstrate optimal substructure →	3. Show optimal soln can be assembled from optimal soln's to sub-problems

Let's see this in action

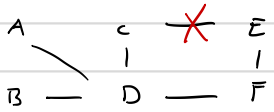
Minimal Spanning Tree (MST)

Motivation: Network of computers, want to connect them all, but minimize cost, each link has a maintenance cost

Instance:



observe: cycles can only hurt you as removing an edge of a cycle will reduce cost



⇒ looking for a spanning tree

↑ undirected, connected, acyclic graph

tree iff $|E| = |V| - 1$

hence, the problem is

Input: Undirected graph $G=(V,E)$, edge weights w_e s.t. $\forall e \in E$, $w_e \in \mathbb{R}$ ← $w_e < 0$ ok

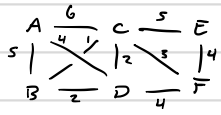
Output: A tree $T=(V,E')$ s.t. 1. $E' \subseteq E$

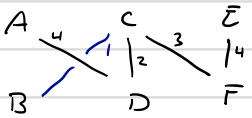
2. E' minimizes $weight(T) = \sum_{e \in E'} w_e$ ← total weight of all edges in the tree

Solving via greedy approach: Need all 3 ingredients above.

1. Cast as series of 1-problem sub-choices

heuristic: add lightest adjacent edge that doesn't produce a cycle



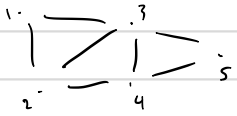


weight(T) = 14

2. Show greedy choice property

Will prove this, but first need some definitions

Def'n For an undirected graph $G=(V,E)$, a cut is a partition of V.

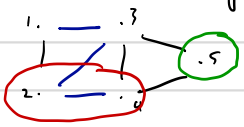
eg 

a cut: $(\{1,2,3\}, \{4,5\})$ ← for $S \subseteq V$, write a cut as $(S, V-S)$

Recall →

Def'n a cut $(S,V-S)$ respects $A \subseteq E$ a subset of edges if $\forall (u,w) \in A$, either $u \in S$ and $w \in S$ or $u \in V-S$, $w \in V-S$

i.e. no edges of A cross the cut

eg 

Say A = set of blue lines,

cut $(\{2,4\}, V-\{2,4\})$ does not respect A (since edge (3,2) crosses the cut)

cut $(\{5\}, V-\{5\})$ does respect A

Theorem: Say A is a subset of edges for some MST of $G=(V,E)$. Let $(S,V-S)$ be a cut that respects A. Let e be the lightest edge that crosses the cut $(S,V-S)$. Then $A \cup \{e\}$ is part of some MST.

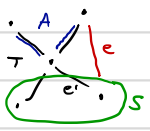
To prove this, we will need

Lemma: Any connected, undirected graph $G=(V,E)$ is a tree iff $|E| = |V| - 1$

Proof: See text (pg 129)

Proof of theorem: By assumption A is a subset of edges of some MST T. If e is part of T, nothing to prove. If $e \notin T$, we will construct a new MST T' w/ $e \in T'$.

Add e to T, this will create a cycle. This cycle must have some edge e' that crosses the cut $(S,V-S)$. Since e is the lightest edge across the cut, $w(e) \leq w(e')$. Let $T' = T \cup \{e\} - \{e'\}$. Now T' is a tree (by the lemma since we didn't change the edge count).



Also,

$$\text{weight}(T') = \text{weight}(T) - w(e') + w(e)$$

and hence $\text{weight}(T') \leq \text{weight}(T)$ ← T is an MST and so $\text{weight}(T') = \text{weight}(T)$

Thus T' is an MST. □

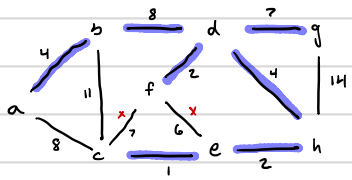
How does this help us? It guarantees that the greedy choice (the edge e in the proof) is part of an optimal sol'n (T' in the proof), so the greedy choice is safe.

3. Even better, it shows optimal substructure: each time we apply the thm, we are guaranteed to be part of an MST, so when no more edges to add, we will have an MST!

Turning this into an algorithm, we get

Kruskal's Algorithm

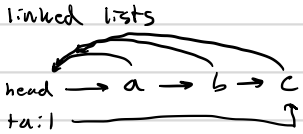
```
def Kruskal_MST(G, w)        # connected, undirected graph; edge weights
    A = ∅
    for all v ∈ G.V
        make_set(v)          # singleton set w/ v in it
    G.E = G.E sorted in non-decreasing order by w
    for all (u,v) ∈ G.E
        if find_set(u) ≠ find_set(v)   # find the set v belongs to
            A = A ∪ {(u,v)}
            union(u,v)       # merge the sets containing u and v
    return A
```

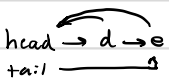


Running time

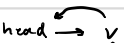
Depends on how we implement make_set, find_set, union

Could use linked lists





make_set(v)



find_set(v) := return head

union(u,v) = concatenate smaller onto larger, update heads

union(a,d) →

length of smaller list

Need to update $O(\min(L_a, L_d))$ heads

If updating single head is $O(1)$, how much does union(·,·) cost in Kruskal? Head pointer of x only gets updated when it's the smaller list ⇒ list doubles in size ⇒ each node head pointer updated @ most $\log_2|V|$ times

So total running time: $O(|V|)$ for all make_set's ($\log E \approx \log V$)

$O(|E| \log |E|) = O(|E| \log |V|)$ to sort edges

$O(|E| \log |V|)$ for all the unions

∴ total is

$$O(|V| + |E| \log |V|)$$

But! we can do better!